

Lezione n.18

PROTOCOLLI P2P

EPIDEMICI

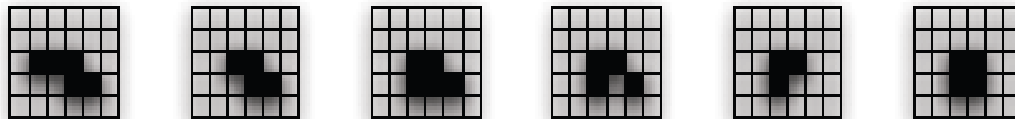
materiale didattico:

Van Steen, GRAPH THEORY AND COMPLEX NETWORKS
articolo sulla pagina del corso

Laura Ricci
11/5/2010

ALGORIMI EPIDEMICI

- Diversi fenomeni naturali, come quello della diffusione dei virus, possono essere utilizzati come paradigma per lo sviluppo di algoritmi distribuiti
- La diffusione di un virus, può essere modellata mediante una semplice regola
 - una persona infetta che viene a contatto con una non infetta ha una probabilità dell'85% di infettare la persona sana
 - a partire da questa regole è possibile definire modelli matematici che descrivono la **diffusione del virus**
- Esempio di un semplice algoritmo epidemico: the **Game of Life**



Nascita: una cella vuota che ha tre celle vicine occupate, genera un nuovo individuo

Morte : un individuo che ha 0 o 1 vicini muore di solitudine. Un individuo che ha da 4 ad 8 vicini muore per soffocamento

Sopravvivenza: Un individuo che ha 2 o 3 vicini sopravvive

ALGORITMI EPIDEMICI

- Diversi fenomeni possono essere descritti mediante un **algoritmo epidemico**, o **algoritmo di gossip**
 - Il 'pettegolezzo' umano
 - La diffusione dei virus
 - Fenomeni naturali come l'incendio di una foresta
 - Worms, virus in ambienti informatici
- Caratteristiche generali:
 - estrema semplicità
 - robustezza
 - efficienza nella diffusione di un'informazione
- Applicazioni in computer science
 - Virus, worms
 - definizione di protocolli p2p

ALGORITMI EPIDEMICI: STRUTTURA GENERALE

Algoritmo epidemico

- si ispirano a comportamenti che si presentano in natura
- esiste una **popolazione** composta da un insieme di entità comunicanti
- esiste un insieme di semplici regole che definiscono come ogni entità può diffondere una informazione alle altre
- ogni entità ha esclusivamente una **visione locale** dell'ambiente
- ogni entità si può trovare in uno dei seguenti stati
 - **infettabile** (**susceptible**); l'entità non è ancora venuta a conoscenza della informazione da diffondere, ma è in grado di riceverla
 - **infettata**; l'entità è venuta a conoscenza di una specifica informazione e può diffonderla, rispettando un insieme di regole
 - **rimossa**; l'entità è venuta a conoscenza di una specifica informazione, ma non la diffonde

ALGORITMI EPIDEMICI: CLASSIFICAZIONE

- **Susceptible-Infective (SI)**. ogni unità è inizialmente **infettabile**, quando riceve una informazione aggiornata diventa **infettata** e rimane tale finchè l'intera popolazione viene infettata
- **Susceptible-Infective-Susceptible(SIS)**: ogni unità può decidere autonomamente di interrompere la diffusione dell'informazione, prima che la popolazione sia completamente infettata. L'unità può riprendere in seguito la diffusione dell'informazione
 - esempio: una unità verifica che le ultime k unità contattate sono già state infettate, e quindi può decidere che l'informazione è obsoleta ed interromperne la diffusione, per poi riprenderla in seguito
- **Susceptible-Infective-Removed(SIR)**: come nel caso precedente una unità può decidere di interrompere la diffusione di un'informazione. In questo caso non può riprendere successivamente la diffusione dell'informazione.
 - esempio: una unità che non è riuscita a diffondere l'informazione negli ultimi x turni, decide che l'intera popolazione è stata infettata ed interrompe la diffuizione dell'informazione

STRATEGIE DI DIFFUSIONE DELL'INFORMAZIONE

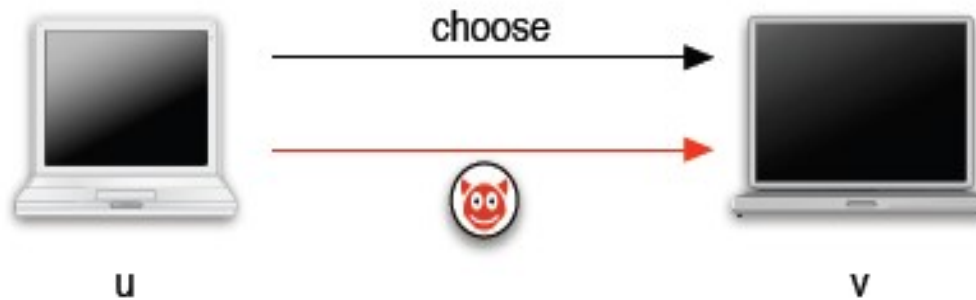
Modelli di comunicazione per algoritmi epidemici SI

- la computazione avviene secondo una **sequenza di cicli**
- ad ogni ciclo, i nodi si contattano a coppie, diffondendo l'informazione

Modelli di interazione: supponiamo che una entità u scelga in modo casuale uniforme un'altra unità v

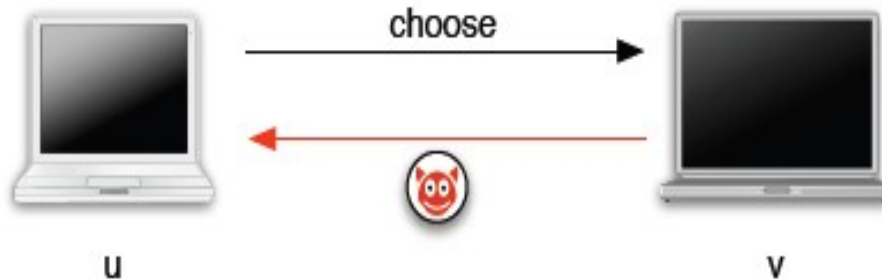
- **Push:** u diffonde a v l'informazione aggiornata, u **infetta** v
- **Pull:** u **chiede** a v se possiede una informazione aggiornata, cioè accade se v è **infetta**. In questo caso u viene infettato da v
- **PushPull:** Se una delle due unità (u o v) è infetta (possiede un'informazione aggiornata), entrambe le unità diventano infette

STRATEGIA PUSH

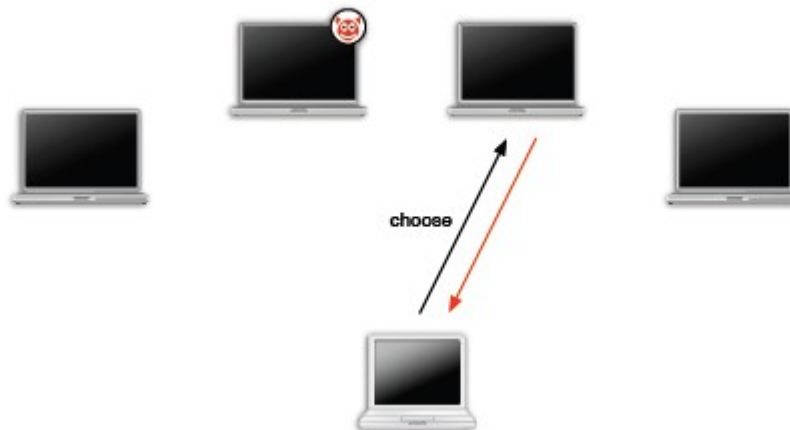


- All'inizio, solo pochi nodi sono infetti e la probabilità di scegliere un nodo non infetto è alta
- Successivamente la probabilità di trovare un nodo non infetto diminuisce

STRATEGIA PULL

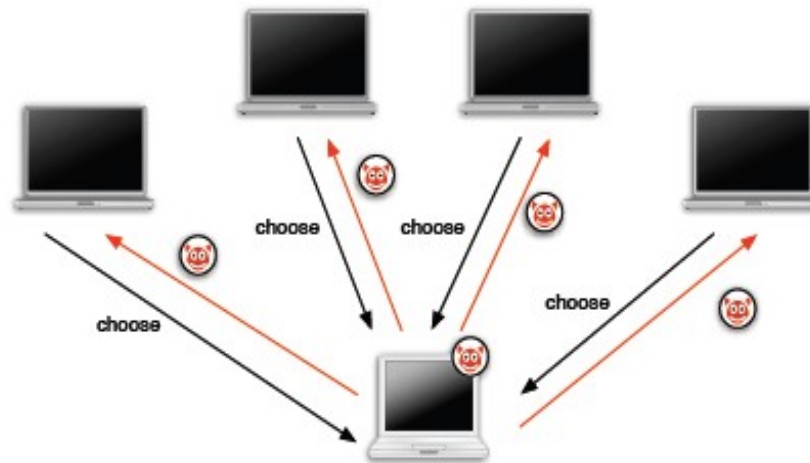


- all'inizio la probabilità di essere infettati è bassa
- non esiste garanzia che la diffusione dell'informazione inizi al primo ciclo

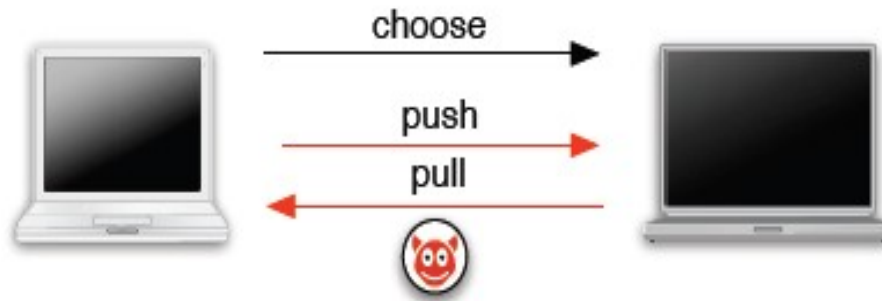


STRATEGIA PULL

- Esiste anche la possibilità che l'intera popolazione venga infettata in un solo ciclo



STRATEGIA PUSH-PULL



- unisce i vantaggi degli approcci precedenti
- all'inizio dominano le operazioni di push, perchè il numero di unità non infettate è maggiore di quelle infettate, nell'ultima parte quelle di pull, perchè è più probabile individuare nodi infettati
- la probabilità che un nodo non infetto contatti un nodo infetto cresce ad ogni ciclo, mentre quella che un nodo infetto contatti uno non infetto diminuisce ad ogni round

ALGORITMI EPIDEMICI PER DATA BASE REPLICATI

- L'idea degli algoritmi epidemici è stata originariamente proposta per gestire la consistenza in database replicati
- Si considerino N nodi, ciascuno memorizza un insieme di oggetti
 - Ad ogni oggetto O è associato un nodo primario che è l'unico a generare gli aggiornamenti su O
 - Ad ogni aggiornamento generato per un oggetto O è associato un timestamp
 - $val(O,S)$ indica il valore di O nel nodo S
 - $T(O,S)$ indica il timestamp del valore di O nel nodo S
- L'aggiornamento generato dal nodo primario deve essere diffuso a tutte le altre le copie
- **Anti-entropia:** algoritmi epidemici il cui scopo è quello di 'riconciliare' le diverse viste del database corrispondenti a sue diverse repliche

ANTI ENTROPIA: APPROCCI PUSH/PULL

Supponiamo che durante la fase di anti-entropia, il nodo S contatti il nodo T

- **Push:** $T(O,T) < T(O,S) \Rightarrow \text{val}(O,T) := \text{val}(O,S)$
- **Pull:** $T(O,T) > T(O,S) \Rightarrow \text{val}(O,S) := \text{val}(O,T)$
- **Push-Pull:** S e T scambiano i loro valori e memorizzano il valore più aggiornato
- Algoritmo epidemico: ogni nodo sceglie periodicamente un altro nodo, in maniera uniforme tra tutti i nodi del sistema per scambiare gli aggiornamenti.
- Ogni aggiornamento viene propagato in $O(\log(N))$ cicli, dove N è il numero di nodi del sistema

ANTI ENTROPIA: APPROCCIO PUSH

- Consideriamo una popolazione di N nodi. Supponiamo che esista un unico nodo che produce aggiornamenti e li propaga mediante un **algoritmo epidemico SI**
- Valutiamo la probabilità p_{i+1} che un nodo T **non risulti infettato** al ciclo $i+1$
 - Approccio **push**:
 - T non era infettato al ciclo i
 - nessun nodo infetto lo ha contattato al ciclo i per scambiare gli aggiornamenti

$$p_{i+1} = p_i \left(1 - \frac{1}{N}\right)^{N(1-p_i)} \approx p_i e^{-1}$$

- Sia S_N il primo ciclo in cui tutta la popolazione è infettata, cioè $p_i=1$. Vale che

$$S_N = \log(N) - \ln(N) + O(1)$$

- La diffusione della informazione avviene in tempo logaritmico

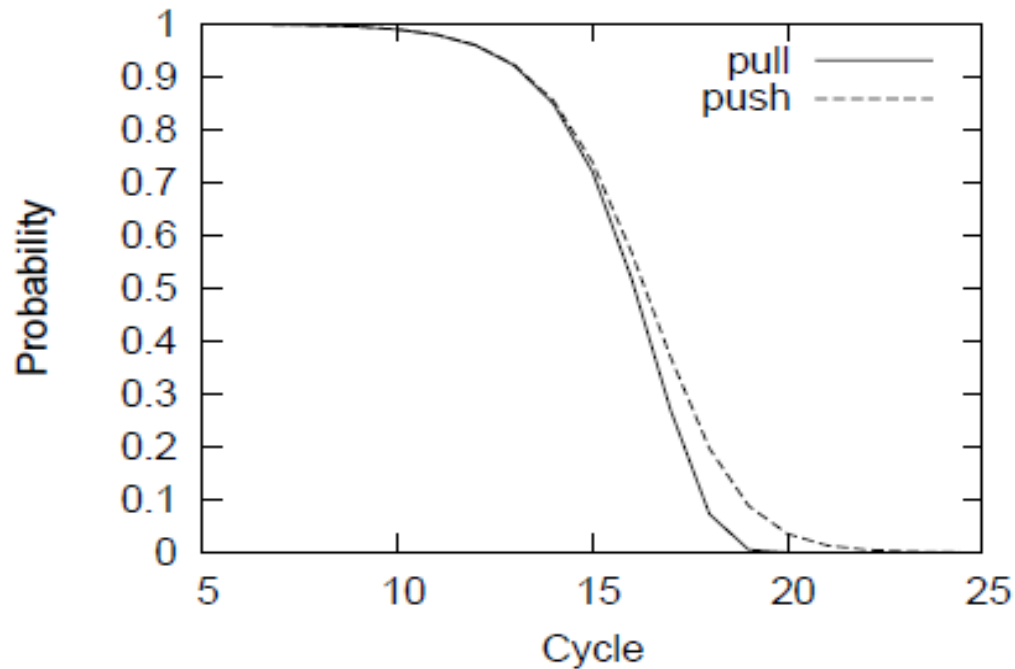
ANTI ENTROPIA: APPROCCI PULL PUSH/PULL

- Consideriamo una popolazione di N nodi. Supponiamo che esista un unico nodo che produce aggiornamenti e li propaga mediante un **algoritmo epidemico SI**
- Valutiamo la probabilità p_{i+1} che un nodo T **non risulti infettato** al ciclo $i+1$
 - Approccio **pull**:
 - T non era infettato al ciclo i
 - contatta un nodo non infettato al ciclo i

$$p_{i+1} = (p_i)^2$$

- la velocità di diffusione dipende da p_i
- il numero di messaggi richiesti è $O(N \log \log N)$
- il risultato è valido anche per l'approccio push/pull
- modelli matematici più complessi per i modelli SIS e SIR

PUSH VS. PULL



Probabilità che una entità non risulti infettata al ciclo I

Velocità di diffusione maggiore per l'approccio pull

ALGORITMI EPIDEMICI

Modello di riferimento

- un insieme dinamico di nodi distribuiti
- ogni nodo partecipa al protocollo epidemico
 - i nodi possono unirsi/lasciare la rete dinamicamente
 - i nodi possono fallire in un qualsiasi istante
- comunicazione:
 - ogni nodo può comunicare con un sottoinsieme degli altri nodi di cui conosce l'indirizzo IP
 - i messaggi possono essere persi. Il protocollo deve funzionare anche in presenza di un alta percentuale di perdita di messaggi
- gli algoritmi epidemici 'classic' si basano su una assunzione importante:
 - ad ogni ciclo dell'algoritmo un nodo P può selezionare un nodo Q scelto in **modo casuale uniforme** tra l'insieme di tutti i nodi che partecipano al protocollo

PEER SAMPLING SERVICE

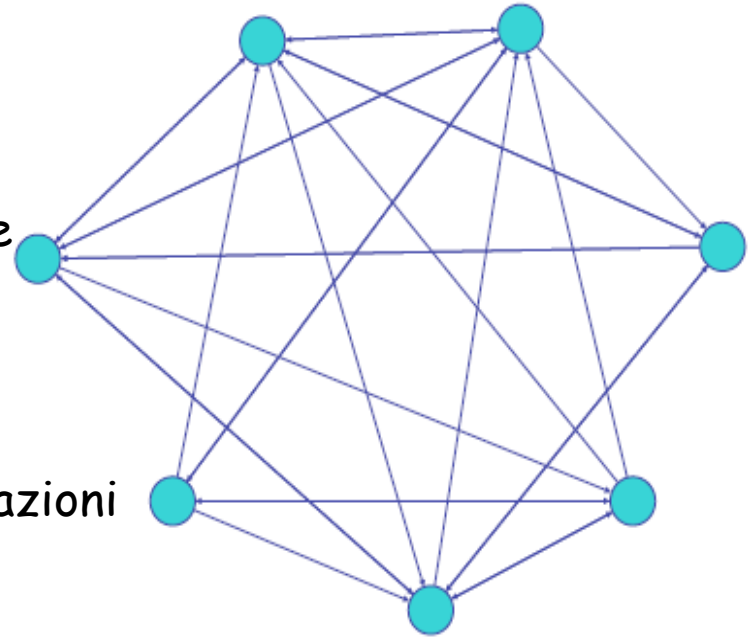
- in ogni algoritmo epidemico
 - **scelta vicini**: un peer sceglie in modo casuale un insieme di vicini con cui scambiare l'informazione (vicini da **infettare**)
 - **gossip**: **scambia** un insieme di informazioni con tali vicini. Esempio: propagazione degli update per riconciliare le repliche in un database
- **Peer sampling service**: **getPeer()**
 - **input**: l'insieme P dei peer che eseguono l'algoritmo epidemico
 - **output**: insieme di peer scelti **casualmente in modo uniforme** tra quelli di P
- In un ambiente P2P il Peer Sampling Service deve garantire
 - **scalabilità**
 - **accuratezza in presenza di alta dinamicità**: l'insieme dei peer restituiti dovrebbe essere scelto tra quelli **attualmente presenti** sulla rete
 - **indipendenza**: peer diversi dovrebbero ottenere 'campioni' indipendenti

PEER SAMPLING SERVICE DISTRIBUITO

- Un servizio di campionamento accurato richiede la presenza di un server centralizzato che registri la presenza dei peer e restituisca un campione di peer scelto in **modo casuale uniforme**
 - questa soluzione non è realizzabile in un ambiente P2P perchè non scalabile
- In un ambiente P2P il servizio di sampling può essere implementato mediante un **algoritmo epidemico**
 - ogni peer possiede una **propria vista della rete**
 - i peer si **scambiano frequentemente le proprie vista** (gossip) cambiando continuamente quindi la propria visione della rete
 - il sampling service restituisce un sottoinsieme dei peer presenti nella vista del peer che invoca la `getPeer()`
 - questa soluzione consente di approssimare accuratamente la soluzione in cui il 'campione' restituito dal servizio è scelto dall'insieme di tutti i peer della rete
- Il servizio di sampling viene utilizzato dagli algoritmi epidemici ai livelli superiori

PEER SAMPLING MEDIANTE ALGORITMI EPIDEMICI

- ogni nodo possiede una propria vista contenente C vicini
- per ogni vicino:
indirizzo IP
informazioni necessarie per implementare il servizio di campionamento
- ogni nodo contatta periodicamente un vicino nella propria vista e scambia con esso informazioni relative alla propria vista (gossip)
- i nodi coinvolti nel gossip aggiornano la propria vista in base alle informazioni ricevute
- la vista di un nodo cambia continuamente: **overlay altamente dinamico**



STRUTTURA GENERALE DI ALGORITMI DI GOSSIP

Active thread

```
selectPeer (&Q) ;  
selectToSend (&bufs) ;  
sendTo (Q, bufs) ;  
  
receiveFrom (Q, &bufr) ;  
selectToKeep (p_view, bufr) ;
```

Passive thread

```
receiveFromAny (&P, &bufr) ;  
selectToSend (&bufs) ;  
sendTo (P, bufs) ;  
selectToKeep (p_view, bufr) ;
```

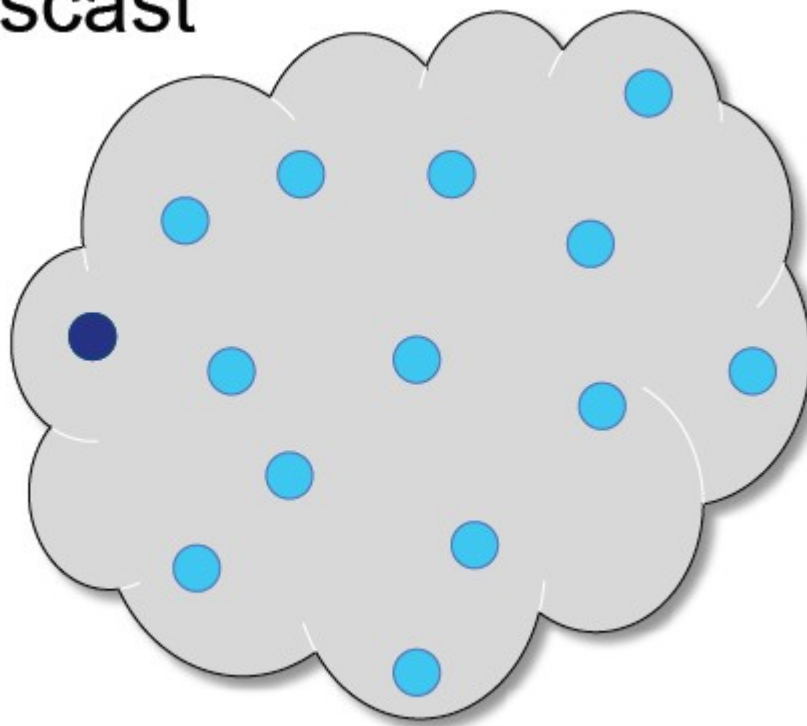
- **SelectPeer** : seleziona in modo casuale un peer dalla vista locale
- **SelectToSend** : seleziona alcune entrate dalla vista locale
- **SelectToKeep** :
 - aggiunge le informazioni ricevute alla vista locale,
 - elimina i duplicati e seleziona un sottoinsieme della vista risultante che definisce la nuova vista locale

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

- Decrittore di un peer: identificatore del peer + timestamp
- **SelectPeer:** seleziona un peer in modo random un peer dalla vista locale
- **SelectToSend:** restituisce tutti i descrittori appartenenti alla vista locale + un descrittore del peer associato ad un timestamp = 0
- **SelectToKeep:**
 - unisce la vista locale con quella ricevuta
 - restituisce i descrittori più recenti, cioè quelli con timestamp maggiore




UN ESEMPIO: IL PROTOCOLLO NEWSCAST

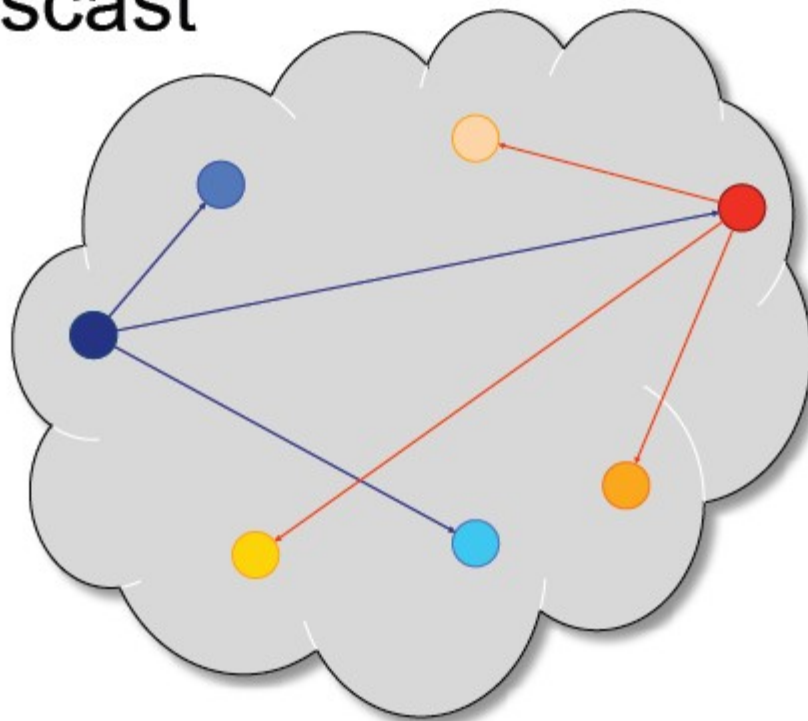
Newscast



UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast



ID & Address	Time stamp
	9
	12
	16

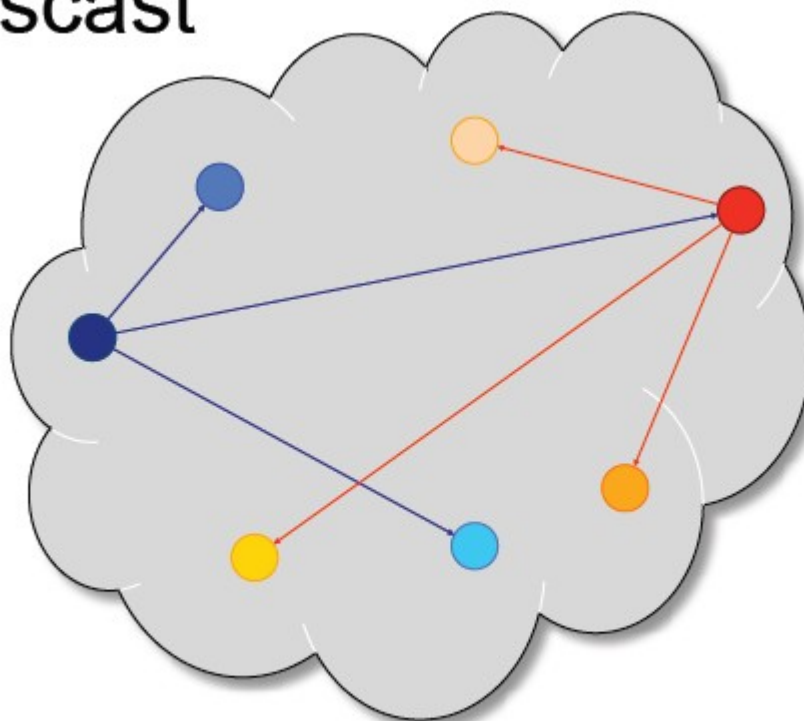


ID & Address	Time stamp
	7
	10
	14

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
	9
	12
	16






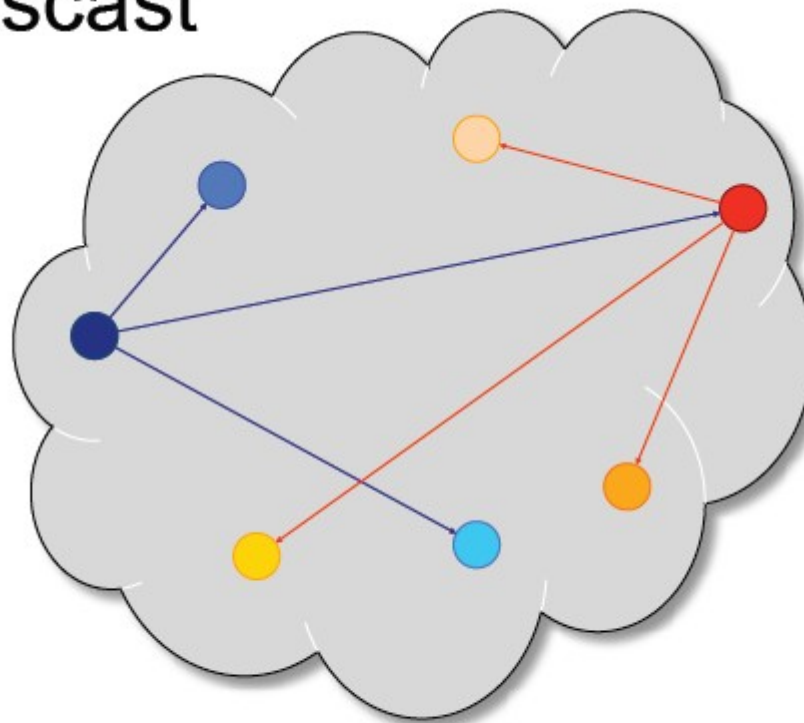
ID & Address	Time stamp
	7
	10
	14

1. Pick random peer from my view

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

News

ID & Address	Time stamp
	9
	12
	16






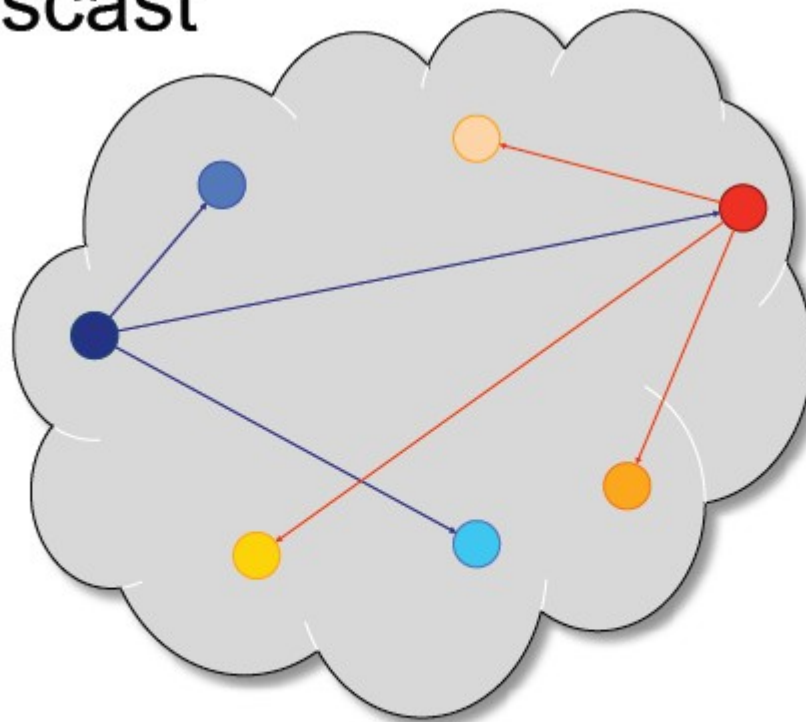
ID & Address	Time stamp
	7
	10
	14

1. Pick random peer from my view

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
	9
	12
	16






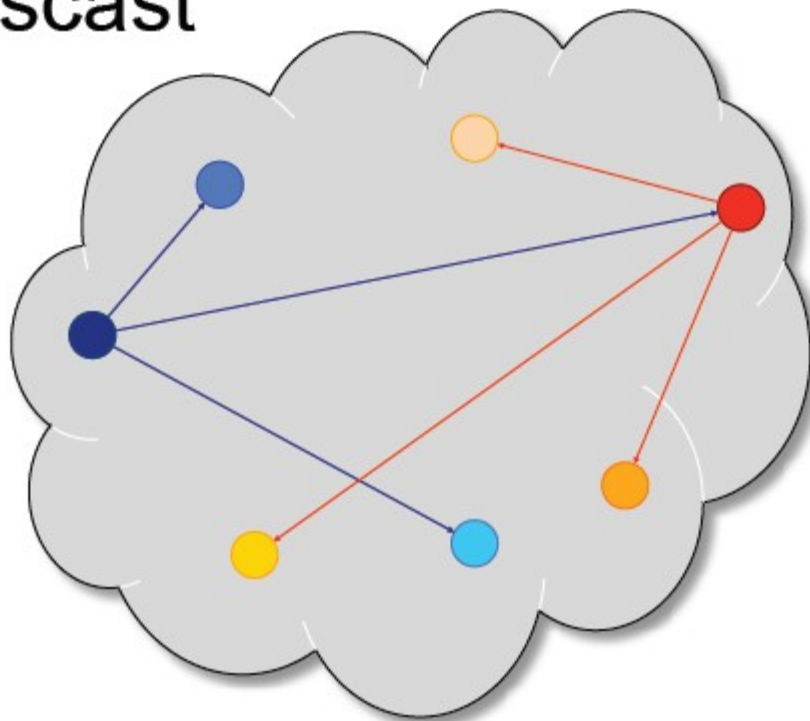
ID & Address	Time stamp
	7
	10
	14

1. Pick random peer from my view
2. Send each other view + own fresh link




UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
	9
	12
	16



ID & Address	Time stamp
	7
	10
	14




	9
	12
	16




	20
---	----


1. Pick random peer from my view
2. Send each other view + own fresh link

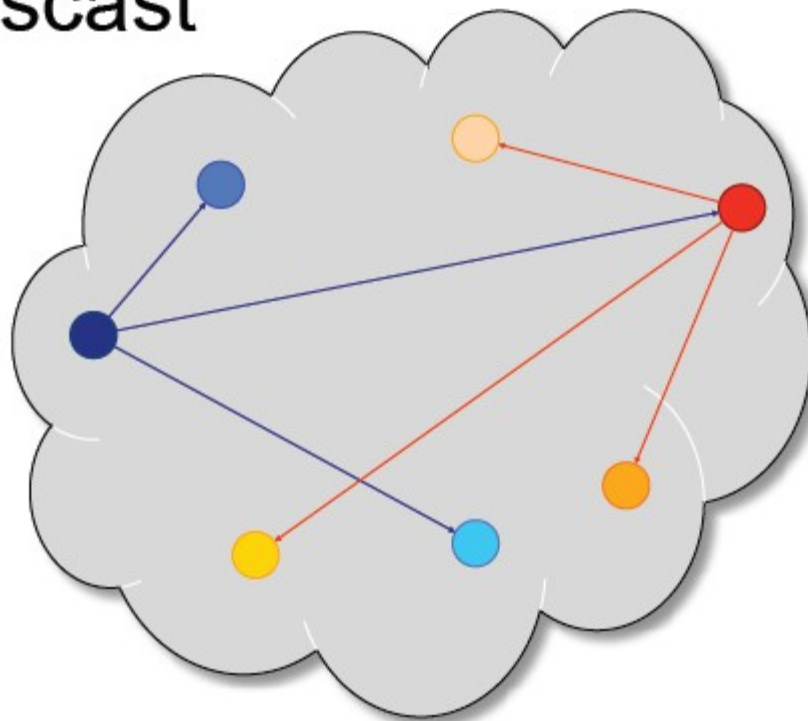
UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
	9
	12
	16

	7
	10
	14

	20
---	----



ID & Address	Time stamp
	7
	10
	14








	9
	12
	16

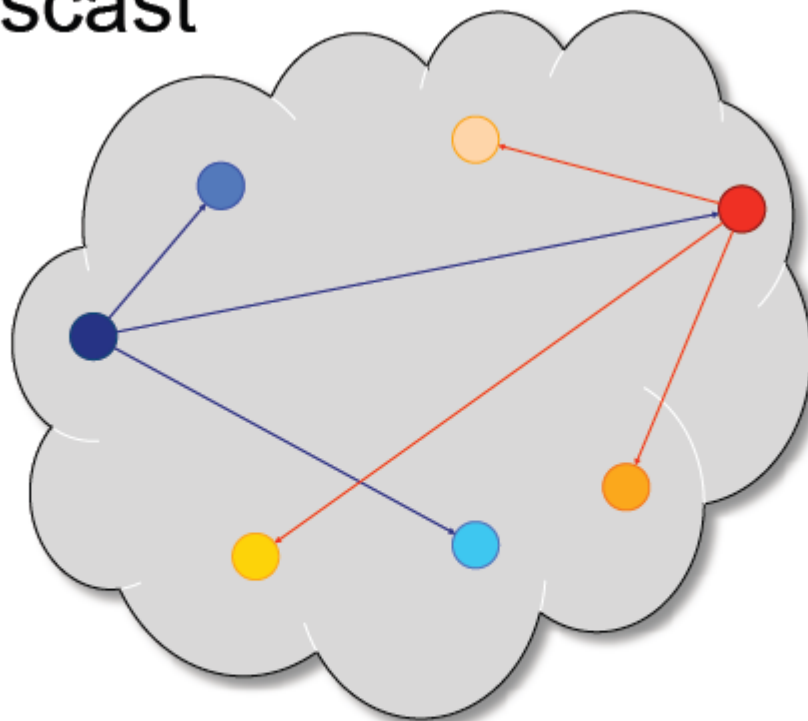
	20
---	----

1. Pick random peer from my view
2. Send each other view + own fresh link

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
	9
	12
	16
	7
	10
	14
	20










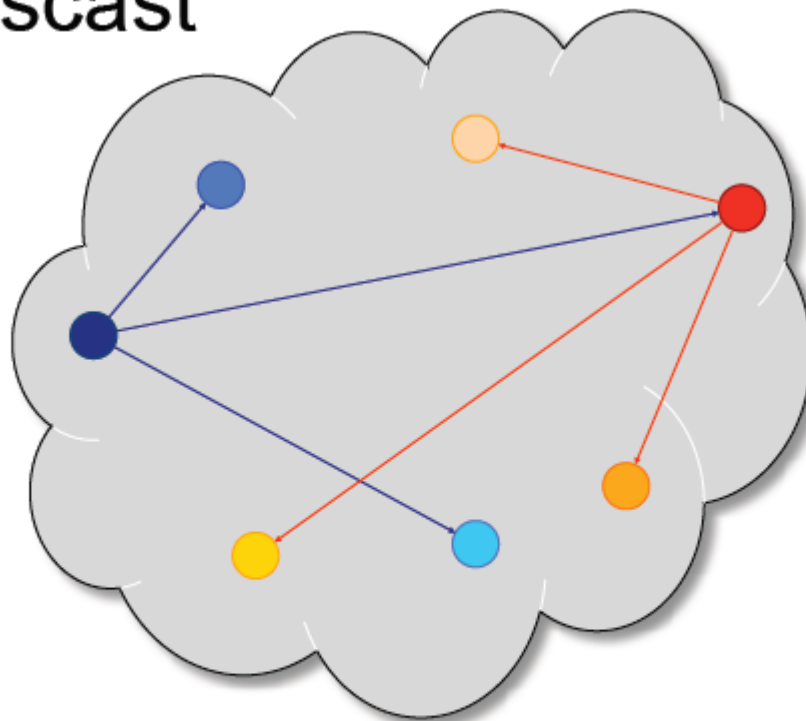
ID & Address	Time stamp
	7
	10
	14
	9
	12
	16
	20

1. Pick random peer from my view
2. Send each other view + own fresh link
3. Keep c freshest links (remove own info, duplicates)

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
	9
	12
	16
	7
	10
	14
	20



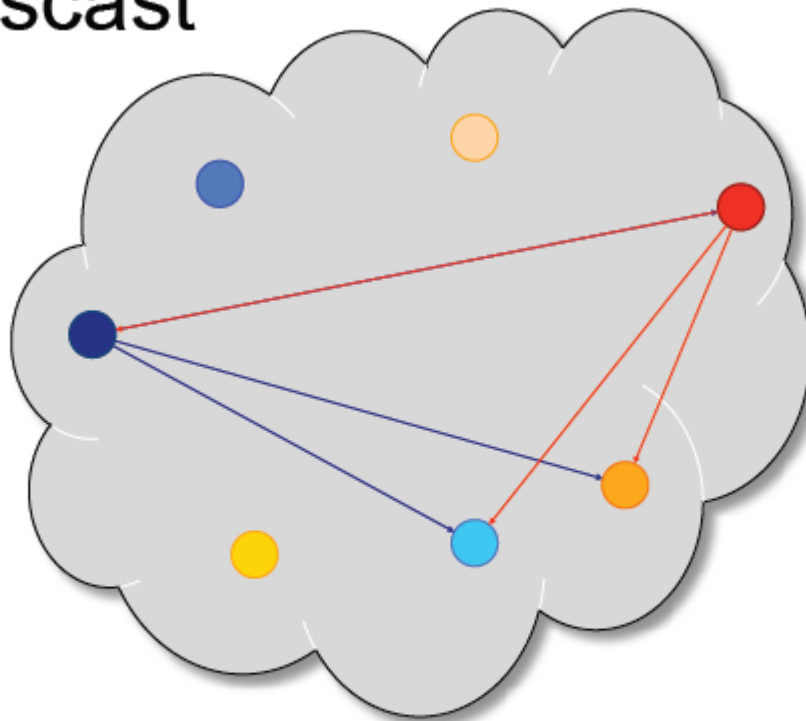
ID & Address	Time stamp
	7
	10
	14
	9
	12
	16
	20

1. Pick random peer from my view
2. Send each other view + own fresh link
3. Keep c freshest links (remove own info, duplicates)

UN ESEMPIO: IL PROTOCOLLO NEWSCAST

Newscast

ID & Address	Time stamp
Orange	14
Light Blue	16
Red	20



ID & Address	Time stamp
Orange	14
Light Blue	16
Dark Blue	20

1. Pick random peer from my view
2. Send each other view + own fresh link
3. Keep c freshest link (remove own info, duplicates)

ALGORITMI DI GOSSIP: STRUTTURA GENERALE

- N numero di nodi della rete, ciascun nodo identificato da un indirizzo
- ogni nodo possiede una visione 'parziale' della rete: una lista locale contenente *c* descrittori
- descrittore del nodo = $\langle \text{indirizzo nodo, età} \rangle$
- operazioni definite sulla vista parziale del nodo:
 - **SelectPeer ()** : restituisce un peer delle vista
 - **permute ()** : shuffle casuale dei peer appartenenti alla vista locale
 - **IncreaseAge ()** : aggiunge 1 all'età di ogni descrittore
 - **Append ()** : aggiunge un insieme di elementi in coda alla vista
 - **RemoveDuplicates ()** : rimuove i duplicati (solito indirizzo), mantenendo le versioni più nuove
 - **removeOldItems (n)** : rimuove gli n descrittori 'più vecchi'
 - **removeHead (n)** : rimuove i primi n descrittori
 - **removeRandom (n)** : rimuove n decsrittori scelti in modo random

ACTIVE THREAD (UNO PER NODO)

```
do forever
  wait(T time units) // T is called the cycle length
   $p \leftarrow \text{view.selectPeer}()$  // Sample a live peer from the current view
  if push then // Take initiative in exchanging partial views
     $\text{buffer} \leftarrow (\langle \text{MyAddress}, 0 \rangle)$  // Construct a temporary list
     $\text{view.permute}()$  // Shuffle the items in the view
    move oldest H items to end of view // Necessary to get rid of dead links
     $\text{buffer.append}(\text{view.head}(c/2))$  // Copy first half of all items to temp. list
    send buffer to  $p$ 
  else // empty view to trigger response
    send (null) to  $p$ 
  if pull then // Pick up the response from your peer
    receive  $\text{buffer}_p$  from  $p$ 
     $\text{view.select}(c, H, S, \text{buffer}_p)$  // Core of framework – to be explained
   $\text{view.increaseAge}()$ 
```

ACTIVE THREAD

- effettua uno shuffle casuale della vista, dopo aver aggiunto alla vista il proprio descrittore
- sposta i descrittori più vecchi in coda alla vista
- selezione i primi $c/2$ elementi dalla vista e li inserisce in un buffer da inviare al partner
 - **osservazione importante:** i primi $c/2$ elementi della vista sono inviati al partner
- **H = healing:** determina quanto il protocollo risulta 'aggressivo' nella eliminazione di links ritenuti obsoleti

PASSIVE THREAD (UNO PER NODO)

do forever

receive buffer_p from p // *Wait for any initiated exchange*

if pull then // *Executed if you're supposed to react to initiatives*

buffer \leftarrow ($\langle \text{MyAddress}, 0 \rangle$) // *Construct a temporary list*

view.permute() // *Shuffle the items in the view*

move oldest H items to end of view // *Necessary to get rid of dead links*

buffer.append(view.head($c/2$)) // *Copy first half of all items to temp. list*

send buffer to p

view.select(c, H, S, buffer_p) // *Core of framework – to be explained*

view.increaseAge()

VIEW SELECTION

- `view_select(c,H,S,bufferp)` restituisce la nuova 'vista locale' del peer
- Parametri
 - `c`: lunghezza della vista parziale
 - `H(Healing)` : numero degli elementi più vecchi spostati in fondo alla lista
 - `S(Swapped)`: numero di elementi da eliminare dalla testa della lista.
Controlla quanti degli elementi scambiati con il partner vengono eliminati dalla vista

```
method view.select( c, H, S, bufferp )  
  view.append(bufferp) // expand the current view  
  view.removeDuplicates() // Remove by duplicate address, keeping youngest  
  view.removeOldItems( min(H,view.size-c) ) // Drop oldest, but keep c items  
  view.removeHead( min(S,view.size-c) ) // Drop the ones you sent to peer  
  view.removeAtRandom(view.size-c) // Keep c items (if still necessary)
```

STRATEGIE DI SELEZIONE DEI PEER

- **SelectPeer()** restituisce un peer scelto in modo random dalla vista corrente
- Scelte possibili
 - **Head:** sceglie il peer con timestamp maggiore, cioè il peer analizzato più di recente
 - offre poche opportunità di individuare nuovi nodi
 - **Rand:** sceglie un peer in modo casuale
 - **Tail:** sceglie il peer con timestamp minore, cioè quello analizzato meno di recente

PROPAGAZIONE DELLE VISTE

- **Push:** il peer invia un insieme di descrittori al peer selezionato
- **Pull:** il peer riceve i descrittori dai nodi selezionati
- **PushPull:** il peer ed il peer selezionato si scambiano i descrittori

NOTA:

i risultati sperimentali dimostrano che la strategia Pull, anche se diffonde più rapidamente l'informazione non consente di ottenere risultati soddisfacenti

- un nodo non ha l'opportunità di iniettare nella rete il suo descrittore
 - contatta gli altri e riceve le loro informazioni, ma non propaga il suo descrittore
- deve aspettare di essere contattato dagli altri
 - la perdita di connessioni in entrata isola il nodo dalla parte rimanente della rete
- in generale, si considerano solamente strategie di tipo **push/pushpull**

SELEZIONE DI UNA NUOVA VISTA

Parametri critici in `select(c,H,S,buffer)`: c, H, S

Si assuma, per semplicità, c pari

- dopo aver aggiunto alla propria vista i descrittori appartenenti alla vista del vicino, il numero dei descrittori nella vista locale è $c+c/2$. (c vecchia vista + $c/2$ descrittori ricevuti dal partner)
- $[H > c/2] \equiv [H = c/2]$, poiché c è la dimensione minima della vista
- $[S > c/2 - H] \equiv [S = c/2 - H]$, poiché c è la dimensione minima della vista
- la rimozione casuale dei descrittori avviene solo se $S < c/2 - H$
- In deve valere $0 \leq H \leq c/2$ e $0 \leq S \leq c/2 - H$
- Scelte possibili
 - **Blind**: $H=0, S=0$ i descrittori selezionati per la nuova vista sono scelti in modo random
 - **Healer (guaritore)**: $H=c/2, S=0$, si selezionano i descrittori più recenti
 - **Swapper** : $H=0, S=c/2$, si eliminano i descrittori scambiati con il vicino

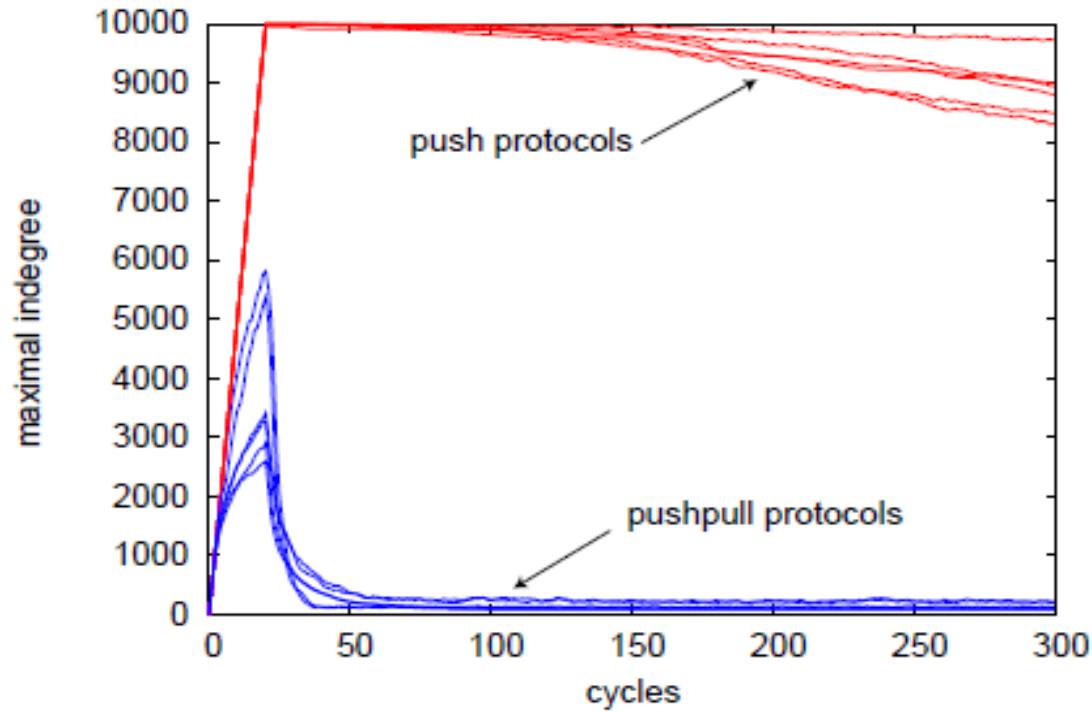
SELEZIONE DI UNA NUOVA VISTA

- Healer:
 - eliminando i descrittori 'più vecchi' si cerca di eliminare i links a peer che non sono più presenti sulla rete
- Swapper
 - i primi $c/2$ descrittori della lista sono quelli scambiati con il partner
 - dopo aver eliminato dalla vista i descrittori più vecchi, si eliminano i primi S descrittori
 - S = swap, il parametro controlla il numero di descrittori scambiati tra i due peer
 - Il parametro S controlla la priorità data ai descrittori ricevuti dal partner
 - un valore alto di S corrisponde ad una probabilità alta di inserire i descrittori ricevuti nella nuova vista
 - un valore basso di S consente di mantenere nella propria vista molti dei valori scambiati con l'altro peer. Come conseguenza, le viste dei due peer **diventano simili**

ANALISI DELL'OVERLAY

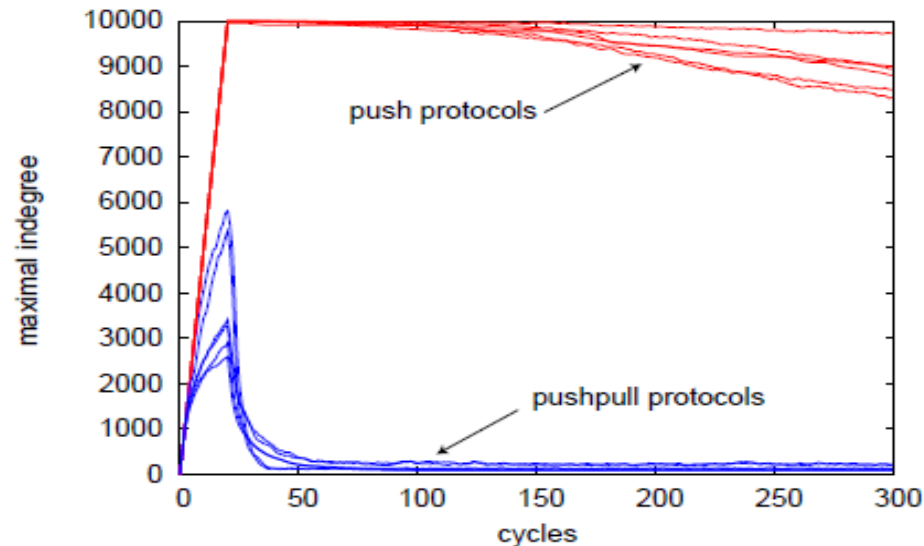
- Analisi dell'overlay generato dall'esecuzione del protocollo
 - il grafo risultante è un **grafo random**?
- Analisi effettuata simulando il protocollo epidemico su una rete di 10000 nodi con c (dimensione della vista) uguale a 30
- Utilizzato il **simulatore Peersim**
- Si considerano tre diverse situazioni iniziali:
 - **Growing**: si inizia con un solo nodo X . Ad ogni ciclo di simulazione si aggiungono 500 nodi. Ognuno dei nuovi nodi all'inizio conosce solo X
 - **Lattice**: Si parte da un overlay in cui i nodi sono connessi ad anello.
 - **Random**: la vista di ogni nodo è inizializzata con nodi scelti in modo casuale uniforme dall'insieme di tutti i nodi
- Osservazione: la strategia Push converge lentamente e produce spesso overlay partizionati nello scenario growing. Di conseguenza si utilizza una strategia push/pull

PUSH VS. PUSH/PULL: INDEGREE



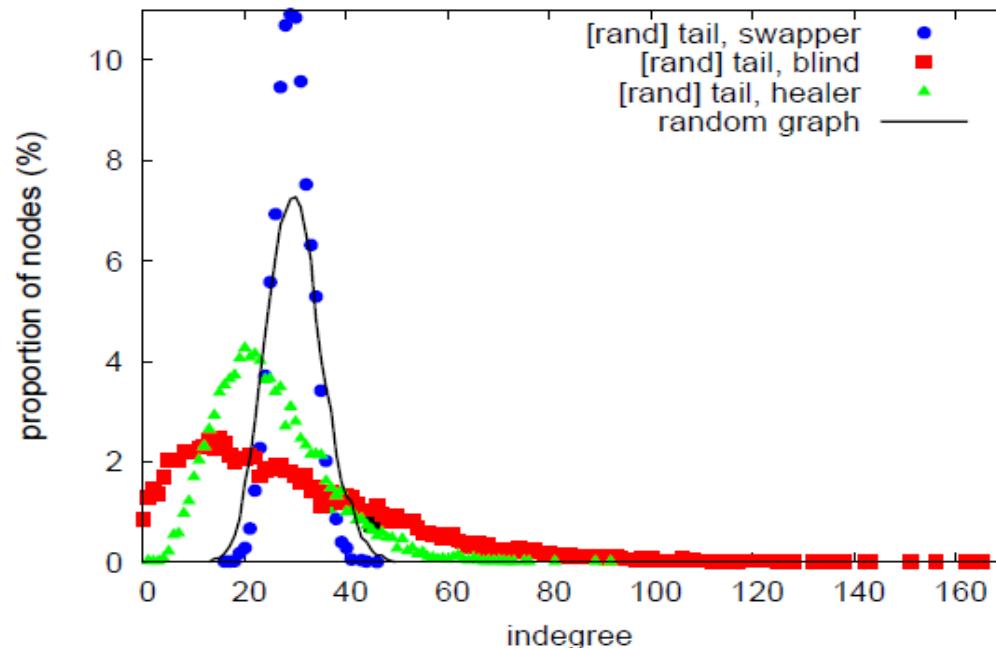
- il grafico mostra l'andamento del massimo indegree in uno scenario di tipo
- Growing: il processo di crescita termina al ciclo 20
- push/pull bilancia immediatamente la distribuzione dei gradi dei nodi
- push non riesce a bilanciare la distribuzione

PUSH VS. PUSH/PULL: INDEGREE



- l'indegree massimo è quello del nodo di bootstrap
- nella strategia **push**
 - quando un nuovo nodo n si inserisce nella rete, esso contatta il nodo di bootstrap e invia il proprio descrittore a tale nodo
 - il descrittore di n viene rapidamente eliminato dalla vista del nodo di bootstrap, perchè nuovi nodi lo contattano continuamente
 - pochi nodi contattano n , il descrittore del bootstrap node rimane nella vista di n

PUSHPULL: INDEGREE



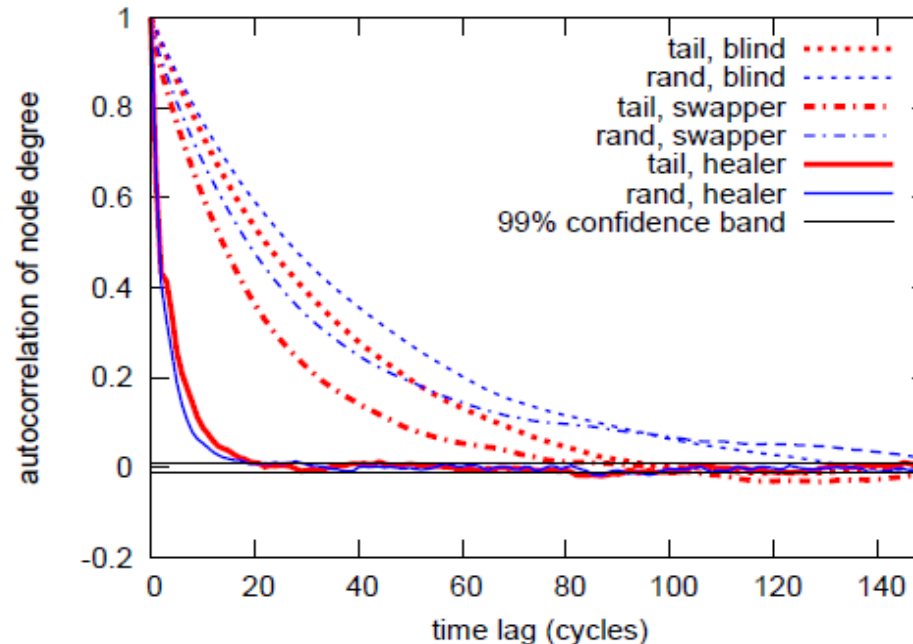
- Analisi della distribuzione degli indegree dei nodi, strategia pushpull
- Swapper bilancia meglio l'indegree rispetto alla strategia blind
- la distribuzione di swapper è migliore anche rispetto a quella ottenuta con un random graph
- la strategia blind presenta 'una lunga coda': molti nodi con indegree alto. Conseguenza negativa sul bilanciamento del carico

INDEGREE: ANALISI DINAMICA

- L'indegree di un nodo cambia durante il tempo
- Quanto rapidamente avviene questo cambio?
- Siano d_1, \dots, d_k , gli indegree per un nodo fissato analizzati considerando K cicli consecutivi di simulazione
- La **correlazione** tra coppie di indegree separati da k cicli di simulazione indica che 'similitudine' c'è tra gli indegree di un nodo in diversi cicli
- La correlazione può essere espressa come:

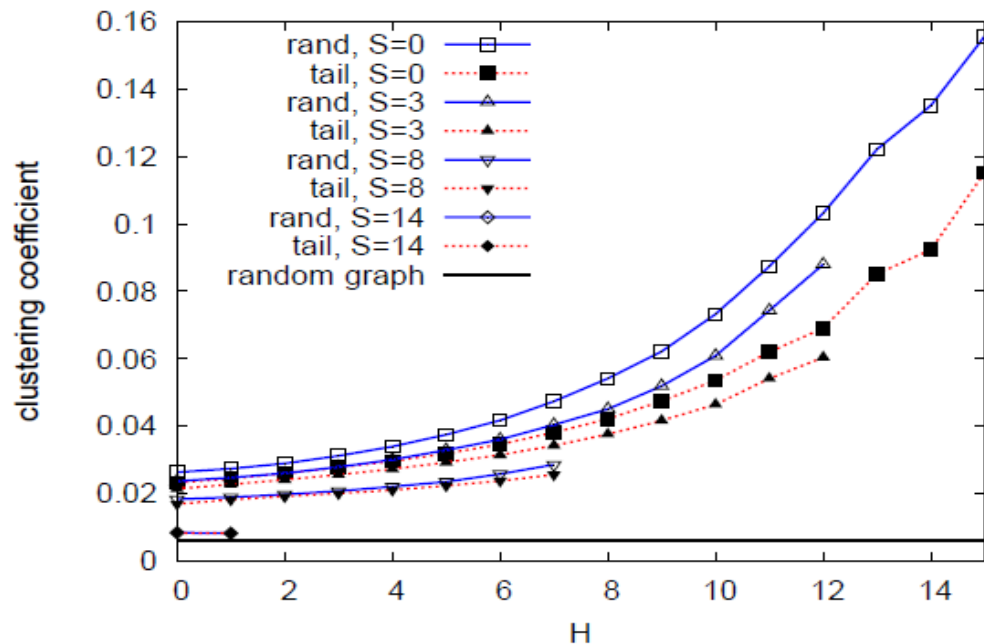
$$r_k = \frac{\sum_{j=1}^{K-k} (d_j - \bar{d})(d_{j+k} - \bar{d})}{\sum_{j=1}^K (d_j - \bar{d})^2}$$

INDEGREE: ANALISI DINAMICA



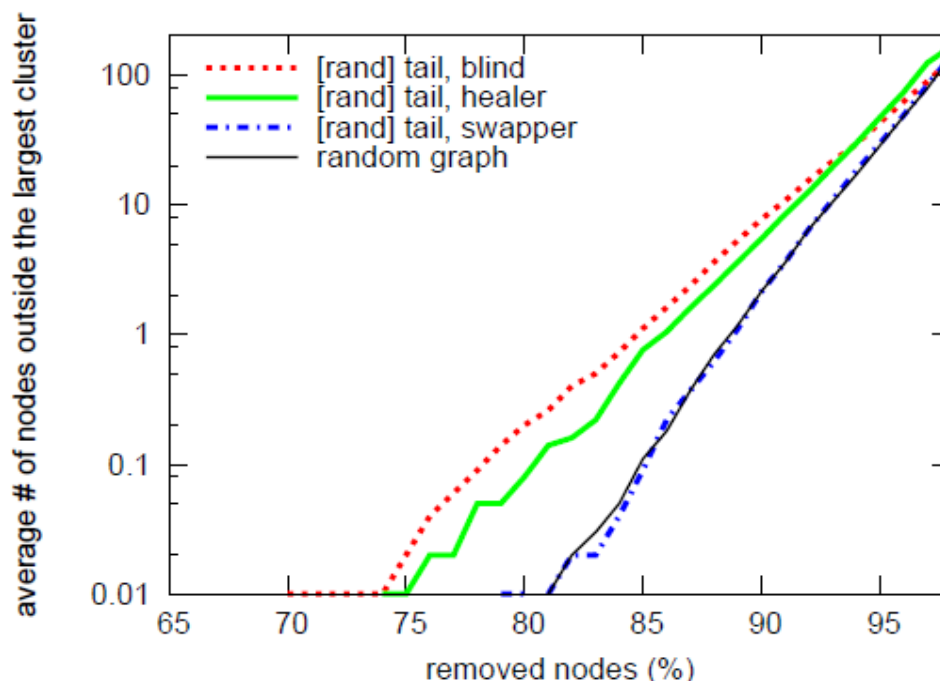
- strategia healer: il grado di un nodo varia rapidamente, non è possibile fare previsioni sul suo grado dopo 20 cicli
- per le altre strategie il grado del nodo cambia molto meno rapidamente ed è possibile trovare una correlazione a distanza di 80-100 cicli
situazione sfavorevole dal punto di vista del bilanciamento del carico: un nodo con un indegree alto può rimanere tale per molti cicli

COEFFICIENTE DI CLUSTERIZZAZIONE



- il coefficiente di clusterizzazione è controllato principalmente da H ed aumenta all'aumentare di H
- se H è alto, le viste scambiate coincidono in gran parte, poiché vengono scartati i descrittori più vecchi e mantenuti quelli ricevuti dal partner
- valori alti di S diminuiscono il coefficiente di clusterizzazione

COMPORTAMENTO IN CASO DI FALLIMENTI

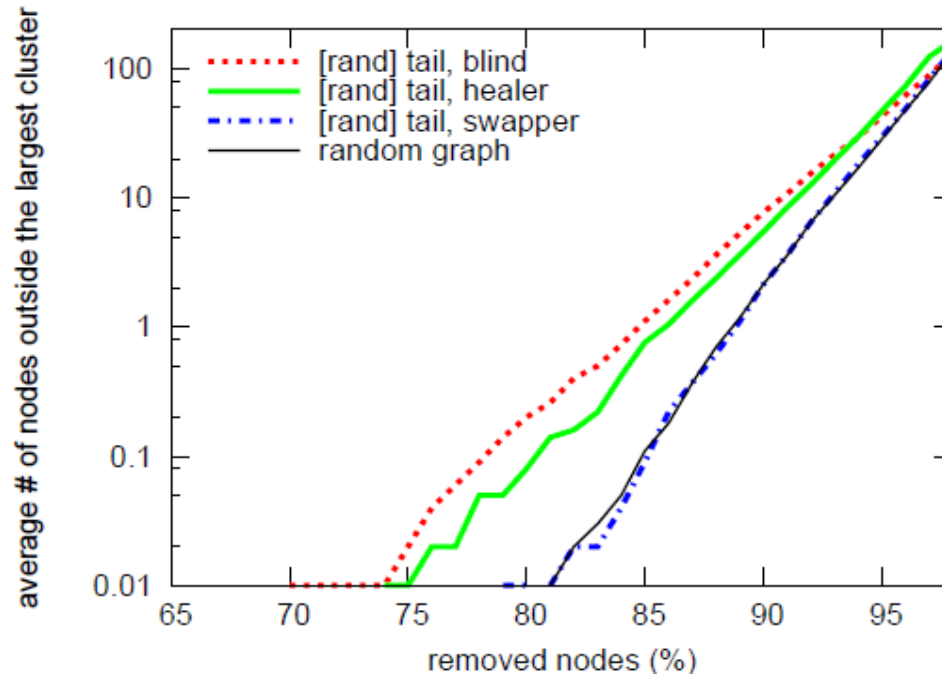


Self-healing: capacità della rete di 'auto ripararsi' in seguito a guasti di natura grave

L'esperimento effettuato:

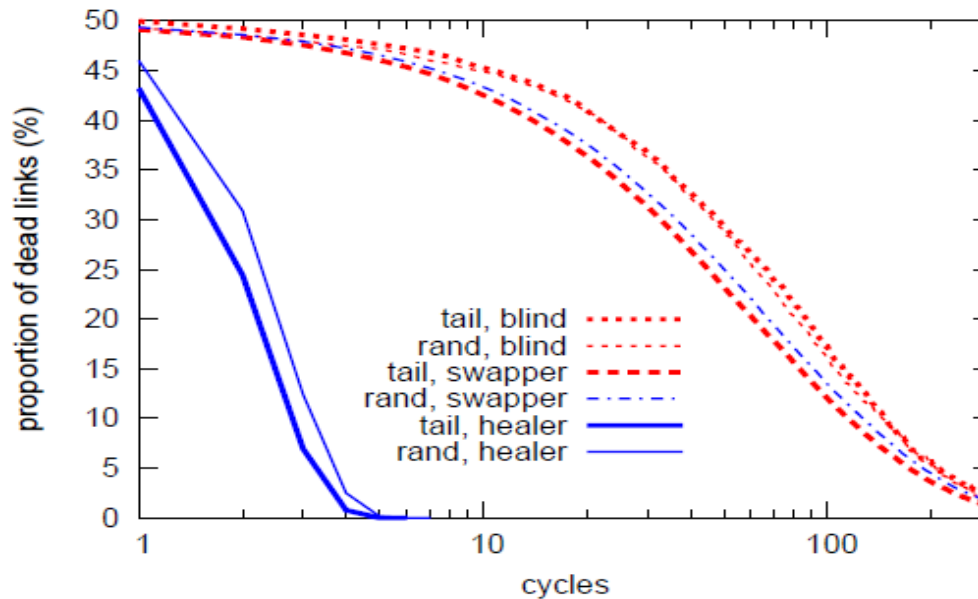
- si eseguono 300 cicli di gossiping per ottenere un overlay 'stabile'
- si rimuove un alto insieme di nodi, scelti in modo random e si analizza la topologia risultante

COMPORTAMENTO IN CASO DI FALLIMENTI



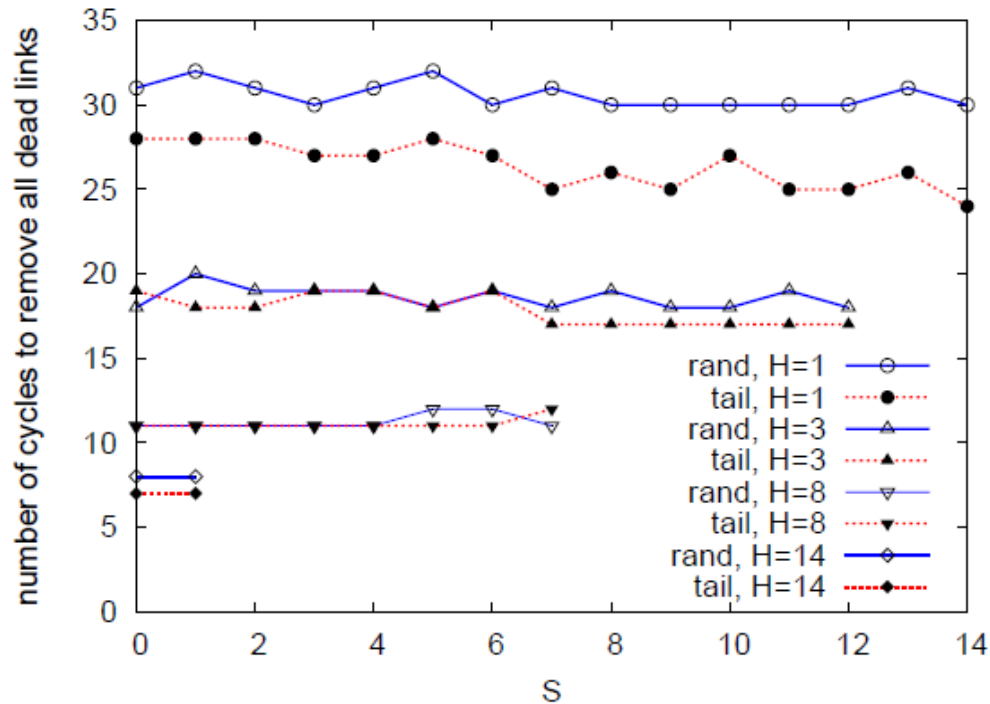
- la rete non subisce partizionamenti se si rimuove meno del 67% dei nodi
- il grafico mostra la percentuale di nodi che non appartengono al cluster più grande (nodi sparsi)
- anche se avviene il partizionamento la maggior parte dei nodi rimane nel cluster più grande
- comportamento molto simile a quello di un grafo random

DEAD LINKS



- Esperimento: si provoca il fallimento del 50% dei nodi al ciclo 300 della simulazione
in media metà dei nodi appartenenti alla vista locale di un peer non fanno più parte della rete
- **Dead links:** descrittori corrispondenti a nodi che non appartengono alla rete
- il grafico mostra il grado di **self-healing** della rete: quanto la rete
- è capace di rimuovere i dead links dalle viste dei nodi

DEAD LINKS



- Il grado di self healing della rete è controllato dal parametro H
- a valori maggiori di H corrisponde una maggiore capacità di Self Healing della rete

ALGORITMI EPIDEMICI: APPLICAZIONI

- Algoritmi di aggregazione (media, max, min, numero di elementi sulla rete)
- Costruzione di overlays
 - Proximity networks
 - Mantenimento di DHT
 - Reti sociali
- Algoritmi di ordinamento distribuiti
- Implementazione distribuita di algoritmi ispirati alla biologia

CONCLUSIONI

- La strategia push-pull risulta migliore delle strategie che utilizzano solo push o solo pull
- **Self Healing:** buona capacità della rete di reagire a 'guasti catastrofici'
- L'eliminazione dei descrittori più vecchi risulta una buona soluzione per aumentare il grado di self healing della rete
- Comportamento complessivo simile a quello di un grafo random
- Le caratteristiche dell'overlay possono essere determinate definendo valori opportuni dei parametri H ed S
- La strategia swapper tende a mantenere un buon bilanciamento degli indegree tra i nodi